The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC501-SW-42

VxWorks Device Driver

32 Channel 16-bit ADC PMC

Version 4.0.x

User Manual

Issue 4.0.0

June 2012

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com www.tews.com

TPMC501-SW-42

VxWorks Device Driver

32 Channel 16-bit ADC PMC

Supported Modules:
TPMC501

Issue	Description	Date
1.0	First Issue	April 15, 1999
1.1	New PCI Configuration	July 16, 1999
1.2	Support for x86 target	June 19, 2000
1.3	General Revision	November 24, 2003
1.3.1	Release.txt added, Issue layout changed	March 8, 2005
2.0.0	New driver startup functions, ChangeLog.txt added to file list, description for tpmc501PciInit changes	January 22, 2007
2.0.1	Function read(): description of parameter *maxbytes* changed	January 11, 2008
3.0.0	VxBus Support, SMP Support and API description added	January 18, 2011
3.0.1	Corrections	November 17, 2011
4.0.0	New API functions, Chapter “Debugging and Diagnostic” added, Chapter “Basic I/O Functions” removed	June 10, 2012

Table of Contents

1 Introduction

1.1 Device Driver

The TPMC501-SW-42 VxWorks device driver software allows the operation of the supported PMC conforming to the VxWorks I/O system specification.

The TPMC501-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x release and mandatory for VxWorks 64-bit and SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API).

Both drivers invoke a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TPMC501-SW-42 device driver supports the following features:

- Execute AD conversion and read data
- Choosing gain, channel, input interface for read
- Correction of input data with board-specific calibration data
- Support of ADC sequencer mode
- Configurable sequencer cycle time, input FIFO size, and channel parameters
- Sequencer read with wait and no wait option

The TPMC501-SW-42 supports the modules listed below:

TPMC501	32(16) Channel - 16-bit ADC	(PMC)

To get more information about the features and use of supported devices it is recommended to read the manuals the supported modules listed below.

TPMC501 User Manual
TPMC501 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC501-SW-42':

TPMC501-SW-42-4.0.0.pdf	PDF copy of this manual
TPMC501-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TPMC501-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TPMC501-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc501':

tpmc501drv.c	TPMC501 device driver source
tpmc501def.h	TPMC501 driver include file
tpmc501.h	TPMC501 include file for driver and application
tpmc501api.c	TPMC501 API file
tpmc501api.h	TPMC501 API include file
Makefile	Driver Makefile
40tpmc501.cdf	Component description file for VxWorks development tools
tpmc501.dc	Configuration stub file for direct BSP builds
tpmc501.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tpmc501exa.c	Example application

The archive TPMC501-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc501':

tpmc501drv.c	TPMC501 device driver source
tpmc501def.h	TPMC501 driver include file
tpmc501.h	TPMC501 include file for driver and application
tpmc501pci.c	TPMC501 device driver source for x86 based systems
tpmc501api.c	TPMC501 API file
tpmc501api.h	TPMC501 API include file
tpmc501exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions

2.1 Legacy vs. VxBus Driver

In later VxWorks 6.x releases, the old VxWorks 5.x legacy device driver model was replaced by VxBus-enabled device drivers. Legacy device drivers are tightly coupled with the BSP and the board hardware. The VxBus infrastructure hides all BSP and hardware differences under a well defined interface, which improves the portability and reduces the configuration effort. A further advantage is the improved performance of API calls by using the method interface and bypassing the VxWorks basic I/O interface.

VxBus-enabled device drivers are the preferred driver interface for new developments.

The checklist below will help you to make a decision which driver model is suitable and possible for your application:

Legacy Driver	VxBus Driver
▪ VxWorks 5.x releases ▪ VxWorks 6.5 and earlier releases ▪ VxWorks 6.x releases without VxBus PCI bus support	▪ VxWorks 6.6 and later releases with VxBus PCI bus ▪ SMP systems (only the VxBus driver is SMP safe) ▪ 64-bit systems (only the VxBus driver is 64-bit compatible)

TEWS TECHNOLOGIES recommends not using the VxBus Driver before VxWorks release 6.6. In previous releases required header files are missing and the support for 3rd-party drivers may not be available.

2.2 VxBus Driver Installation

Because Wind River doesn't provide a standard installation method for 3rd party VxBus device drivers the installation procedure needs to be done manually.

In order to perform a manual installation extract all files from the archive TPMC501-SW-42-VXBUS.zip to the typical 3rd party directory *installDir/vxworks-6.x/target/3rdparty* (whereas *installDir* must be substituted by the VxWorks installation directory).

After successful installation the TPMC501 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tpmc501.*

At this point the TPMC501 driver is not configurable and cannot be included with the kernel configuration tool in a Wind River Workbench project. To make the driver configurable the driver library for the desired processor (CPU) and build tool (TOOL) must be built in the following way:

(1) Open a VxWorks development shell(e.g. C:\WindRiver\wrenv.exe -p vxworks-6.9)

(2) Change into the driver installation directory
installDir/vxworks-6.x/target/3rdparty/tews/tpmc501

(3) Invoke the build command for the required processor and build tool with optional VXBUILD argument
make CPU=cpuName TOOL=tool [VXBUILD=xxx]

For Windows hosts this may look like this:

```
C:> cd \WindRiver\vxworks-6.9\target\3rdparty\tews\tpmc501
C:> make CPU=PENTIUM4 TOOL=diab
```

To compile SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

```
C:> make CPU=PENTIUM4 TOOL=diab VXBUILD=SMP
```

To build 64-bit libraries, the argument VXBUILD=LP64 must be added to the command line

```
> make CPU=CORE TOOL=gnu VXBUILD=LP64
```

For 64-bit SMP-enabled libraries a build command may look like this

```
> make CPU=CORE TOOL=gnu VXBUILD="LP64 SMP"
```

To integrate the TPMC501 driver with the VxWorks development tools (Workbench), the component configuration file *40tpmc501.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.9\target\3rdparty\tews\tpmc501
C:> copy 40tpmc501.cdf \Windriver\vxworks-6.9\target\config\comps\vxWorks
```

In VxWorks 6.7 and newer releases the kernel configuration tool scans the CDF file automatically and updates the *CxrCat.txt* cache file to provide component parameter information for the kernel configuration tool as long as the timestamp of the copied CDF file is newer than the one of the *CxrCat.txt*. If your copy command preserves the timestamp, force to update the timestamp by a utility, such as *touch*.

In earlier VxWorks releases the CxrCat.txt file may not be updated automatically. In this case, remove or rename the original *CxrCat.txt* file and invoke the make command to force recreation of this file.

```
C:> cd \Windriver\vxworks-6.7\target\config\comps\vxWorks
C:> del CxrCat.txt
C:> make
```

After successful completion of all steps above and restart of the Wind River Workbench, the TPMC501 driver and API can be included in VxWorks projects by selecting the *"TEWS TPMC501 Driver"* and *"TEWS TPMC501 API"* components in the "hardware (default) - Device Drivers" folder with the kernel configuration tool.

2.2.1 Direct BSP Builds

In development scenarios with the direct BSP build method without using the Workbench or the vxprj command-line utility, the TPMC501 configuration stub files must be copied to the directory *installDir/vxworks-6.x/target/config/comps/src/hwif.* Afterwards the *vxbUsrCmdLine.c* file must be updated by invoking the appropriate make command.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc501
C:> copy tpmc501.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc501.dr \Windriver\vxworks-6.7\target\config\comps\src\hwif

C:> cd \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> make vxbUsrCmdLine.c
```

2.3 Legacy Driver Installation

2.3.1 Include device driver in VxWorks projects

For including the TPMC501-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Extract all files from the archive TPMC501-SW-42-LEGACY.zip to your project directory.

(2) Add the device driver's C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files in the tpmc501 directory can be selected.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.3.2 Special installation for Intel x86 based targets

The TPMC501 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem an MMU mapping entry has to be added for the required TPMC501 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc501pci.c** contains the function *tpmc501PciInit().* This routine finds out all TPMC501 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tpmc501PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
    tpmc501PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TPMC501 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.3.3 BSP dependent adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two ways of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it, using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

Definition	**Description**
USERDEFINED_MEM_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access
USERDEFINED_IO_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access
USERDEFINED_LEV2VEC	The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header)

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.

2.4 System Resource Requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	---	2

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 API Documentation

3.1 General Functions

3.1.1 tpmc501Open

NAME

tpmc501Open() – open a device.

SYNOPSIS

```
TPMC501_DEV tpmc501Open
(
    char        *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TPMC501 device is named “/tpmc501/0”, the second device is named “/tpmc501/1” and so on.

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE     hdl;

/*
** open file descriptor to device
*/
hdl = tpmc501Open(“/tpmc501/0”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device descriptor pointer or NULL if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

3.1.2 tpmc501Close

NAME

tpmc501Close() – close a device.

SYNOPSIS

```
TPMC501_STATUS tpmc501Close
(
    TPMC501_HANDLE    hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE     hdl;
TPMC501_STATUS     result;

/*
** close the device
*/
result = tpmc501Close(hdl);
if (result != TPMC501_OK)
{
     /* handle close error */
}
```

RETURNS

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC501_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.3 tpmc501GetModuleInfo

NAME

tpmc501GetModuleInfo – Get module information data

SYNOPSIS

```
TPMC501_STATUS tpmc501GetModuleInfo
(
        TPMC501_HANDLE                          hdl,
        TPMC501_INFO_BUFFER                     *pModuleInfo
)
```

DESCRIPTION

This function reads module information data such as configured module type, location on the PCI bus and factory programmed correction data.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pModuleInfo

This argument specifies a pointer to the module information buffer.

```
typedef struct
{
        unsigned int        Variant;
        unsigned int        PciBusNo;
        unsigned int        PciDevNo;
        int                 ADCOffsetCal[4];
        int                 ADCGainCal[4];
} TPMC501_INFO_BUFFER;
```

Variant

This parameter returns the configured module variant (e.g. 10 for a TPMC501-10).

PciBusNo, PciDevNo

These parameters specifies the PCI location of this module

ADCOffsetCal[4]

This array returns the factory programmed offset correction value for the different gains. Array index 0 contains the value for gain 1, index 1 contains the value for gain 2 and so forth.

ADCGainCal[4]

This array returns the factory programmed gain correction for the different gains. Array index 0 contains the value for gain 1, index 1 contains the value for gain 2 and so forth.

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE        hdl;
TPMC501_STATUS        result;
TPMC501_INFO_BUFFER   ModuleInfo

result = tpmc501GetModuleInfo(hdl, &ModuleInfo);
if (result != TPMC501_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC501_ERR_INVALID_HANDLE	The specified TPMC501_HANDLE is invalid.

3.2 Device Access Functions

3.2.1 tpmc501SetModelType

NAME

tpmc501SetModelType – configures the TPMC501 board type

SYNOPSIS

```
TPMC501_STATUS tpmc501SetModelType
(
        TPMC150_HANDLE          hdl,
        int                     ModuleType
)
```

DESCRIPTION

This function configures the TPMC501 board type.

This function must be called after initialization of the ADC device, before any other function accesses the device.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

modType

This parameter specifies the model type of the TPMC501. The following model types are supported.

Model type	Description
10	TPMC501-10 --- input range +/- 10V, gains: 1,2,5,10, front panel I/O
11	TPMC501-11 --- input range +/- 10V, gains: 1,2,4,8, front panel I/O
12	TPMC501-12 --- input range 0…10V, gains: 1,2,5,10, front panel I/O
13	TPMC501-13 --- input range 0…10V, gains: 1,2,4,8, front panel I/O
20	TPMC501-20 --- input range +/- 10V, gains: 1,2,5,10, back I/O
21	TPMC501-21 --- input range +/- 10V, gains: 1,2,4,8, back I/O
22	TPMC501-22 --- input range 0…10V, gains: 1,2,5,10, back I/O
23	TPMC501-23 --- input range 0…10V, gains: 1,2,4,8, back I/O

EXAMPLE

```
#include “tpmc501api”

TPMC501_HANDLE     hdl;
TPMC501_STATUS     result;

/*
** tell the driver, this is a TPMC501-10
*/
result = tpmc501SetModelType(hdl, TPMC501_TYPE_10);
if (result != TPMC501_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC501_ERR_INVALID_HANDLE	The specified device handle is invalid
TPMC501_ERR_INVAL	Unsupported or invalid TPMC501 model type specified
TPMC501_ERR_BUSY	The device is busy

3.2.2 tpmc501Read

NAME

tpmc501Read – perform AD conversion and read value

SYNOPSIS

```
TPMC501_STATUS tpmc501Read
(
        TPMC501_HANDLE                                  hdl,
        int                                             channel,
        int                                             gain,
        unsigned int                                    flags,
        int                                             *pAdcVal
)
```

DESCRIPTION

This function starts an AD conversion on a specified input channel and returns the converted value.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

channel

This argument specifies the input channel. Allowed values are 1...32 for single ended interface. If a differential interface is selected (TPMC501_DIFF set in flags) the values 1…16 are allowed.

gain

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5, 10 or 1, 2, 4, 8 depending on the module type.

flags

Set of bit flags that control the AD conversion. The following flags could be OR'ed:

Flag	Meaning
TPMC501_DIFF	If this flag is set the ADC input works in differential mode otherwise in single-ended (default).
TPMC501_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC501 EEPROM.
TPMC501_FAST	If this flag is set the fast (polled) mode will be used. The driver will not use interrupts, instead it will wait in a busy loop until the settling time (if necessary) and the conversion is finished. Conversions using this mode will be handled faster, but the processor executes a busy loop and other tasks will not be handled during the loops.

pAdcVal

This argument points to a buffer where the AD value will be returned.

EXAMPLE

```
#include “tpmc501api”

TPMC501_HANDLE     hdl;
TPMC501_STATUS     result;
int                in_value;

/*
** read AD value from channel 5 with gain = 2
** single endend input, correction enabled, use interrupts
*/
result = tpmc501Read(hdl,
                     5,
                     2,
                     TPMC501_CORR | TPMC501_FAST,
                     &in_value);
if (result != TPMC501_OK)
{
     /* handle error */
}
else
{
     printf(“ADC #5: %d\n”, in_value);
}
```

RETURN VALUE

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC501_ERR_INVALID_HANDLE	The specified device handle is invalid
TPMC501_ERR_BUSY	The device is busy
TPMC501_ERR_INVAL	Invalid parameter specified: invalid channel number, gain or flag specified
TPMC501_ERR_TIMEOUT	The conversion timed out
TPMC501_ERR_CONFIG	TPMC501 model type unknown or not configured

3.2.3 tpmc501StartSequencer

NAME

tpmc501StartSequencer – setup and start sequencer operation

SYNOPSIS

```
TPMC501_STATUS tpmc501StartSequencer
(
      TPMC501_HANDLE          hdl,
      unsigned int            CycleTime,
      unsigned int            NumOfBufferPages,
      unsigned int            NumOfChannels,
      TPMC501_CHAN_CONF       *ChanConf
)
```

DESCRIPTION

This function sets up and starts the sequencer. The setup specifies the channels to be used in sequencer mode and how they will be setup, defining gain, correction and input interface. Additional the sequencer cycle time is defined and depth of the drivers sequencer FIFO will be configured.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

CycleTime

This argument specifies the repeat frequency of the sequencer in 100 µs steps. Each time the sequencer timer reaches the programmed cycle time a new AD conversion of all active channels is started. Valid values are in the range from 100 µs to 6.5535 seconds.

NumOfBufferPages

This argument specifies the number of sample blocks in the ring buffer. A sample block contains the samples of all channels (NumOfChannels) per sequencer cycle.

NumOfChannels

This argument specifies the number of active channels for this job. The maximum number is 32.

ChanConf

This array of channel configuration structures specifies the configuration of the active channels. The channel configuration defines the channel number, the gain and some flags. The ordering of channels in a ring buffer page is the same as defined in this array.

```
typedef struct
{
        unsigned int        ChanToUse;
        unsigned int        gain;
        unsigned int        flags;
} TPMC501_CHAN_CONF;
```

ChanToUse

This parameter specifies the input channel number. Valid channels for single-ended mode are 1…32, for differential mode 1...16.

gain

This Parameter specifies the gain for this channel. Valid gains are 1, 2, 5, 10 for *TPMC501-10/-12/-20/-22* and 1, 2, 4, 8 for *TPMC501-11/-13/-21/-23.*

flags

Set of bit flags that control the AD conversion. The following flags could be OR'ed:

Flag	Meaning
TPMC501_DIFF	If this flag is set the ADC input works in differential mode otherwise in single-ended (default).
TPMC501_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC501 EEPROM.

EXAMPLE

```
#include "tpmc501api.h"

TPMC501_HANDLE     hdl;
TPMC501_STATUS     result;
unsigned int       CycleTime;
unsigned int       NumOfBufferPages;
unsigned int       NumOfChannels;
TPMC501_CHAN_CONF  ChanConf[TPMC501_MAX_CHAN];

CycleTime          = 5000;
NumOfBufferPages   = 100;
NumOfChannels      = 2;

…
```

```
…

ChanConf[0].ChanToUse   = 1;
ChanConf[0].gain        = 1;
ChanConf[0].flags       = TPMC501_CORR;

ChanConf[1].ChanToUse   = 20;
ChanConf[1].gain        = 5;
ChanConf[1].flags       = TPMC501_CORR;
…

// start the sequencer
result = tpmc501StartSequencer(hdl,
                               CycleTime,
                               NumOfBufferPages,
                               NumOfChannels,
                               ChanConf);
if (result != TPMC501_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

Error Code	**Description**
TPMC501_ERR_INVALID_HANDLE	The specified device handle is invalid
TPMC501_ERR_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.
TPMC501_ERR_INVAL	At least one of the parameters is invalid.
TPMC501_ERR_CONFIG	TPMC501 model type unknown or not configured
TPMC501_ERR_NOMEM	Allocating buffer failed

3.2.4 tpmc501StopSequencer

NAME

tpmc501StopSequencer – Stop the sequencer

SYNOPSIS

```
STATUS tpmc501StopSequencer
(
    TPMC501_HANDLE          hdl
)
```

DESCRIPTION

This function stops execution of the sequencer mode on the specified device.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE   hdl;
TPMC501_STATUS   result;

result = tpmc501StopSequencer(hdl);
if (result != TPMC501_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC501_ERR_INVALID_HANDLE	The specified device handle is invalid

3.2.5 tpmc501GetDataBuffer

NAME

tpmc501GetDataBuffer – Get next data block of sequencer samples

SYNOPSIS

```
TPMC501_STATUS tpmc501GetDataBuffer
(
        TPMC501_HANDLE                              hdl,
        unsigned int                                flags,
        int                                         *pData,
        unsigned int                                *pStatus
)
```

DESCRIPTION

This function returns the next available data block in the ring buffer containing ADC data of configured sequencer channels.

If specified the function will return immediately, although there is no data available. If the function should wait for data the function returns immediately if data is already available in FIFO or wait for sequencer cycle completion. The function timeout, if there is an abnormal delay during wait.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

flags

Set of bit flags that control the sequencer read. The following flags could be OR'ed:

Flag	Meaning
TPMC501_NOWAIT	If this flag is set the function will return immediately, although there is no data available. If the flag is not set, the function will wait until data is available.
TPMC501_FLUSH	If this flag is set the sequencer FIFO will be flushed and the function will wait for new data otherwise the function will read the next available data set.

pData

This argument is a pointer to an array of integer items where the converted data of a sequencer cycle will be filled in. The number of channels and the channel configuration was setup with the tpmc501StartSequencer function. The used buffer must be at least big enough to receive on integer value for every enabled sequencer channel.
The first array item [0] belongs to the channel configured by ChanConfig[0], the second array item [1] belongs to the channel configured by ChanConfig[1] and so forth. Please refer to the example application for details.

pStatus

This argument is a pointer to a variable which returns the actual sequencer error status. Keep in mind to check this status before each reading. If status is 0 no error is pending. A set of bits specifies the error condition.

Value	**Description**
TPMC501_BUF_OVERRUN	This bit indicates a ring buffer overrun. The error occurred if there is no space in ring buffer to write the new AD data. In this case the new AD values are dismissed. The sequencer was not stopped.
TPMC501_DATA_OVERFLOW	This indicates an overrun in the sequencer data RAM. The error occurred if the driver is too slow to read the data in time. The sequencer was stopped after this error occurred.
TPMC501_TIMER_ERR	Sequencer timer error (see also TPMC501 hardware manual). The sequencer was stopped after this error occurred.
TPMC501_INST_RAM_ERR	Sequencer instruction RAM error (see also TPMC501 hardware manual). The sequencer was stopped after this error occurred.

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE   hdl;
TPMC501_STATUS   result;
unsigned int     seqStatus;
int              numOfSeqChannels;
int              *pData;

numOfSeqChannels = 2;       /* Two channels used in sequenccer mode */

/* allocate sequence input buffer */
pData = malloc(sizeof(int) * numOfSeqChannels);

…
```

```
…

/* read a set of fresh ADC data */
result = tpmc501GetDataBuffer(hdl, TPMC501_FLUSH, &pData, &seqStatus);
if (result != TPMC501_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC501_ERR_INVALID_HANDLE	The specified TPMC501_HANDLE is invalid.
TPMC501_ERR_TIMEOUT	There the expected wait time has been exceeded.
TPMC501_ERR_NOT_READY	The sequencer is stopped.
TPMC501_ERR_NODATA	The function returned without data
TPMC501_ERR_BUSY	The device is not configured in sequencer mode

4 Legacy I/O System Functions

This chapter describes the legacy driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

The legacy I/O system functions are only relevant for the legacy TPMC501 driver. For the VxBus-enabled TPMC501 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tpmc501PciInit

NAME

tpmc501PciInit – Generic PCI device initialization

SYNOPSIS

void tpmc501PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC501 PCI spaces (base address register) and to enable the TPMC501 device for access.

The global variable *tpmc501Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of tpmc501Status is equal to the number of mapped PCI spaces
0	No TPMC501 device found
< 0	Initialization failed. The value of (tpmc501Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tpmc501PciInit();

tpmc501PciInit();
```

5 Debugging and Diagnostic

The TPMC501 device driver provides a function and debug statements to display versatile information of the driver installation and status on the debugging console.

If the VxBus driver is used, the TPMC501 show routine is included in the driver by default and can be called from the VxWorks shell. If this function is not needed or program space is rare the function can be removed from the code by un-defining the macro INCLUDE_TPMC501_SHOW in tpmc501drv.c

The tpmc501Show function (only if VxBus is used) displays detailed information about probed modules, assignment of devices respective device names to probed TPMC501 modules and device statistics.

If TPMC501 modules were probed but no devices were created it may helpful to enable debugging code inside the driver code by defining the macro TPMC501_DEBUG in tpmc501drv.c.

In contrast to VxBus TPMC501 devices, legacy TPMC501 devices must be created "manually". This will be done with the first call to the tpmc501Open API function.

```
-> tpmc501Show
Probed Modules:
    [0] TPMC501: Bus=4, Dev=1, DevId=0x9050, VenId=0x10b5, Init=OK, vxDev=0x478878

Associated Devices:
    [0] TPMC501: /tpmc501/0

Correction Data:
    /tpmc501/0:
                     gain/offset
        Gain = 1:     248/55
        Gain = 2:     241/57
        Gain = 4/5:   235/62
        Gain = 8/10:  240/70

Device Statistics:
    /tpmc501/0:
        open count = 0
        interrupt count = 3
```